SEGTREE TRANSFORMER: ITERATIVE REFINEMENT OF HIERARCHICAL FEATURES

Zihao Ye[†], **Qipeng Guo**[†‡*], **Quan Gan**[†], **Zheng Zhang**[†§]
[†]AWS Shanghai AI Lab
[‡]Fudan University
[§]New York University Shanghai
{yeziha, gqipeng, quagan, zhaz}@amazon.com

ABSTRACT

The building block of Transformer can be seen as inducing message passing over a complete graph whose nodes correspond to input tokens. Such dense connections make the Transformer data-hungry. Star-Transformer exploits short-term dependencies more heavily by keeping the connections between adjacent tokens but relaying long dependencies via a central node, thereby reducing the number of connections from $O(n^2)$ to $O(n)$. This centralized structure has trouble handling long sentences. This paper proposes Segment Tree Transformer (SegTree-Transformer), a middle ground that organizes input tokens into a tree of word spans, and extends attentions to those spans as well. It yields $O(n \log n)$ connections which greatly improves space and time complexity, and outperforms alternatives in a number of NLP tasks with moderately-sized data.

1 INTRODUCTION

Transformer, a self-attention based model, has achieved many impressive results on NLP tasks, notably machine translation (Vaswani et al., 2017), language modelling (Radford et al., 2018), and text classification (Devlin et al., 2018). However, the heavy architecture of Transformer often requires large-scale training data or otherwise suffers performance loss.

A valid perspective is to view the inner-workings of Transformer as message passing on graph, with input tokens as nodes and attentions as edges (Battaglia et al., 2018). The fully connected nature enables each word to communicate with any other token in exactly one step.

Star-Transformer (Guo et al., 2019) employs a *relay node* centralizes information from all input tokens via attention, then each node attends to the relay as well as its immediate neighbors. Such two-phase message passing on a lightweight star-topology balances the importance of short-range dependencies (in the form of n-grams) and that of long-range dependencies *as a whole*, drastically reduces computation load from $O(n^2)$ to $O(n)$, and achieves better performance with training data of moderate size.

Vanilla Transformer and Star-Transformer can be seen as two ends of a spectrum. In vanilla Transformer, non-local relationships are fully distributed, thereby requiring a large memory footprint and a big training set to prevent overfitting. In contrast, Star-Transformer completely centralizes all non-adjacent dependencies, so the relay node can be over-stretched, especially in long sentences.

In this paper, we explore a middle ground with an architecture called SegTree Transformer (SegTree-Transformer), which incorporates the Segment Tree data structure (De Berg et al., 1997) into Transformer. For a given sentence, SegTree-Transformer constructs a complete binary tree whose leaf nodes map to each of the words, and the intermediate nodes representing the span over all words mapped to its descendant leaves. Each node attends to its neighbors just as in Star-Transformer, but instead of attending to one relay node, it attends to a *minimal* subset of nodes in the tree such that the combined spans covers the entire sentence. The idea is to distribute the load, but not as aggressive as Transformer does. The total computation load is $O(n \log n)$.

[*]Work done during internship at AWS Shanghai AI Lab.

Simplifying the model as SegTree-Transformer does effctively encode the inductive bias of attending more on near neighbors and less on far away ones, an observations made in Khandelwal et al. (2018).

2 Model

2.1 Recap: Transformer

Given a sentence with n input tokens, the Transformer model [1] iteratively computes at step t the d-dimensional representations of each input token $H^t \in \mathbb{R}^{n \times d}$, where H^0 is the token embeddings themselves. The core of a Transformer step is Multi-head Self-Attention (**MSA**), which can be formulated as follows:

$$\begin{gathered}\text{MSA}(H) = [\text{head}_1, \cdots, \text{head}_N]W^O \qquad \text{head}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d}}\right) V_i \\ Q_i = HW_i^Q \qquad K_i = HW_i^K \qquad V_i = HW_i^V\end{gathered} \tag{1}$$

Where N is the number of heads, and W_i^Q, W_i^K, W_i^V, W^O are learnable parameters.

The Transformer then computes H^{t+1} from H^t:

$$Z^t = \text{norm}(H^t + \text{MSA}(H^t)) \qquad H^{t+1} = \text{norm}(Z^t + \text{FFN}(Z^t)) \tag{2}$$

Where norm represents the layer normalization (Ba et al., 2016) and FFN stands for the Position-wise Feed-Forward Networks in Vaswani et al. (2017). Note that each step t gets its own parameters of self-attention and feed-forward network.

2.2 SegTree-Transformer: Graph Construction

Instead of attending over every single token, SegTree-Transformer organizes tokens into spans, and each token attends over either spans or other tokens, thereby reducing the number of nodes to attend. SegTree-Transformer does so by constructing a complete binary tree whose leaf nodes correspond to the input tokens. The internal nodes then correspond to spans covering the words mapped to the descendant leaves. We note that generalizing the idea to arbitrary k-nary tree is possible, as long as the invariants (stated below) is abided to.

Once we map the nodes to word spans, we reconnect the nodes in two directions:

bottom-up A directed edge connects each leaf to all ancestors of the original binary tree.

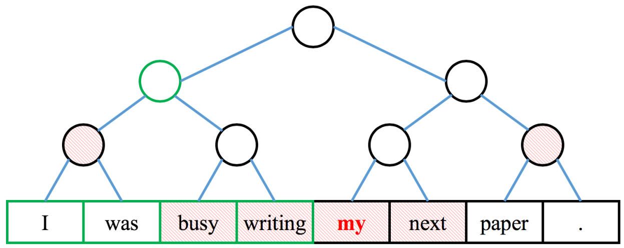
top-down A directed edge connects to each leaf the nodes it is going to attend to. For each leaf, a minimal set of attention context nodes are selected, provided that (1) the leaf itself, and the leaves to its left and right are included, (2) the spans of all selected nodes are disjoint while covering the entire sentence. Figure 1 shows the algorithm of finding such context nodes.

We use $\mathcal{G}$ to denote the whole graph, the number of nodes is $O(2n)$, while the number of edges is $O(n \log n)$. ($O(n \log n)$ for bottom-up edges and $O(n \log n)$ for top-down edges)

2.3 SegTree-Transformer: Message Passing

After graph construction, we alternate between updating the representations of the internal node and the leaves with Transformer-style self-attention. Within a SegTree-Transformer layer, for any given node u, we update its representation h^u to $GSA(\mathcal{G}, h^u)$ ($\mathcal{G}$ the underlying graph, GSA refers to Graph Self-Attention, also seen in Veličković et al. (2017)) as follows:

[1] In the scope of this paper we focus on the *encoder* aspect of the Transformer model, and simply refer that as ”The Transformer model” in short.



function NEIGHBORS(u)
  $L \leftarrow \emptyset, R \leftarrow \emptyset, v \leftarrow u$
  repeat
    $L \leftarrow L \cup \{\text{left}(v) \backslash L\}$
    $R \leftarrow R \cup \{\text{right}(v) \backslash R\}$
    $v \leftarrow \text{parent}(v)$
  until v is root
  return $L \cup \{u\} \cup R$
end function

Figure 1: The binary tree constructed over a sentence. The word in bold would compute self-attention over the shaded nodes(top-down edges), while the internal node with green circle would attend over all its descendant leaves, highlighted with green boxes(bottom-up edges). The algorithm of finding top-down context nodes for a given leaf node is shown on the right, where $u \backslash S$ refers to the node correspond to span $u - u \bigcap S$.

$$A^u = \text{concat}(\{h_v \mid v \in \mathcal{A}(u)\}) \tag{3}$$

$$Q_i^u = H_k W_i^Q \qquad K_i^u = A^u W_i^K \qquad V_i^u = A^u W_i^V \tag{4}$$

$$head_i^u = \text{softmax}\left(\frac{Q_i^u K_i^{uT}}{\sqrt{d}}\right) V_i^u \tag{5}$$

$$\text{GSA}(\mathcal{G}, h_u) = [head_1^u, \cdots, head_N^u] W^O \tag{6}$$

where $\mathcal{A}(u)$ is the predecessors of u in $\mathcal{G}$ and i loops over all attention heads, and $W_{\cdot}^{\cdot}$ and W^O are trainable parameters. Optionally, we can also introduce relative positional embedding in Equation like Shaw et al. (2018) as a variant, which we explain in the Appendix A.2.

Recall that the predecessors of an internal node are all its descendant leaves, while those of a leaf are found in Figure 1. We can see that setting $\mathcal{A}(u)$ in the equations above to all the leaf nodes is equivalent to the vanilla Transformer.

In our experiments, we update all nodes synchronously within a SegTree-Transformer layer. The internal node representations are initialized with all zero, while leaf node representations are initialized with corresponding word embeddings. We can stack multiple SegTree-Transformer layers as in vanilla transformer, where each layer gets its own $W_{\cdot}^{\cdot}$ and W_O. Depending on the downstream tasks, we either take as output of the SegTree-Transformer model the root node representation in the final layer (e.g. in Text Classification), or the concatenation of all the leaf node representations in the final layer (e.g. in Language Modeling).

3 Experiments

We evaluate our model on two NLP tasks on real-world texts. *Language Modeling* measures the quality of leaf node's feature in the tree structure while *Text Classification* measures the quality of root node's feature. As each node may aggregate from a varying number of neighbors, we use Deep Graph Library(DGL) (Wang et al., 2018) to implement the whole model because it can handle such situations easily and efficiently.

3.1 Language Modeling

By masking out edges (u, v) in self-attention where the position node v lies in is ahead of the span node u lies in, our proposed SegTree-Transformer can be applied to language modeling task.

We conduct experiments on two popular datasets: PTB (Marcus et al., 1993) and WikiText2 (Merity et al., 2016). The two datasets consist of collection of long articles, in experiments we divide them into sequences of equal length. To measure our model's ability in long text modeling, we select two different sequence length settings: 70 and 320. To study how the number of layers affect the performance of SegTree-Transformer, we set the number of layers to 3, 4, 5 and 6 respectively.

For all models we fix the embedding size and hidden size to 320, and tie embedding weight with the pre-softmax linear transformation weight, similar to Press & Wolf (2016). The baseline model we select is multi-layer LSTM with dropout on input and between layers. The dropout rate is set to 0.5 for embedding/output layer, and 0.1 for intermediate sublayers in Transformer-based models, 0.3 for all layers in LSTM-based models. For Transformer-based models, the hidden size of position-wise feed forward network is set to 1024.

Model/Datasets	PTB(dev/test)	WikiText-2(dev/test)
LSTM(Sequence Length: 70, 2 Layers)	86.17/82.53	96.99/92.25
LSTM(Sequence Length: 70, 3 Layers)	88.02/83.91	98.13/93.73
LSTM(Sequence Length: 320, 3 Layers)	101.04/98.50	106.59/101.12
Star Transformer(Sequence Length: 70, 3 Layers)	89.92/85.02	113.18/107.61
Vanilla-Transformer(Sequence Length: 70, 3 Layers)	92.72/86.07	100.50/94.72
Vanilla-Transformer(Sequence Length: 70, 4 Layers)	88.53/81.32	97.01/91.56
Vanilla-Transformer(Sequence Length: 70, 5 Layers)	85.85/79.18	93.52/88.53
Vanilla-Transformer(Sequence Length: 70, 6 Layers)	84.73/77.76	94.18/88.38
Vanilla-Transformer(Sequence Length: 320, 3 Layers)	89.58/82.48	95.42/90.46
Vanilla-Transformer(Sequence Length: 320, 4 Layers)	87.07/80.61	89.38/84.53
Vanilla-Transformer(Sequence Length: 320, 5 Layers)	82.66/76.16	82.68/79.11
Vanilla-Transformer(Sequence Length: 320, 6 Layers)	81.21/75.05	**80.30/76.56**
SegTree-Transformer (Sequence Length: 70, 3 Layers)	91.12/83.38 ↓	102.94/97.07 ↑
SegTree-Transformer (Sequence Length: 70, 4 Layers)	86.17/78.71 ↓	98.08/92.88 ↑
SegTree-Transformer (Sequence Length: 70, 5 Layers)	84.87/78.02 ↓	95.50/90.57 ↑
SegTree-Transformer (Sequence Length: 70, 6 Layers)	83.61/77.19 ↓	94.62/89.52 ↑
SegTree-Transformer (Sequence Length: 320, 3 Layers)	85.33/78.69 ↓	92.11/87.45 ↓
SegTree-Transformer (Sequence Length: 320, 4 Layers)	80.56/74.03 ↓	87.96/83.82 ↓
SegTree-Transformer (Sequence Length: 320, 5 Layers)	79.42/72.95 ↓	85.09/81.26 ↑
SegTree-Transformer (Sequence Length: 320, 6 Layers)	**76.63/71.23** ↓	83.07/79.28 ↑

Table 1: Perplexity(the lower the better) on Language Modeling, ↓ indicates SegTree-Transformer outperforms Vanilla-Transformer and ↑ means Vanilla-Transformer performs better.

The result suggests that our model is capable of capturing long-term dependencies (SegTree-Transformer performs better when split the corpus into longer sequences). Unlike LSTM, which performs worse when we increase the number of layers, Transformer and SegTree-Transformer consistently performs better as the number of layers grows. Compared to Vanilla Transformer, our model achieves comparable perplexity(better on PTB but worse on WikiText2) and is more efficient in terms of GPU memory cost(see A.3). Compared to Star Transformer, our model significantly performs better on WikiText-2 which allows for the capture and usage of longer term dependencies (Merity et al., 2016).

3.2 Text Classification

We use SST-1 dataset (Socher et al., 2013) and IMDB dataset (Maas et al., 2011), the previous one having fine-grained labels with 215,154 phrases in 11,855 sentences with average length of 19 and the latter having positive/negative labels on 50,000 multi-sentence reviews with average length 294, to measure the performance of our model on short/long text classification. We use pre-trained GloVe embedding (Pennington et al., 2014) as input features and fixed them during training. The hidden size of all our models are set to 300.

Model	SST-1
SegTree-Transformer	**53.0**
Star Transformer	52.9
Transformer	50.4
Bi-LSTM (Li et al., 2015)	49.8
Tree-LSTM (Socher et al., 2013)	51.0

Table 2: Test Accuracy on SST-1 Dataset

Model	IMDB
SegTree-Transformer	**91.75**
Star Transformer	90.50
Transformer	–
BCN+Char+CoVe (McCann et al., 2017)	91.8

Table 3: Test Accuracy on IMDB dataset

We compare the performance of SegTree-Transformer, Star Transformer and Transformer(we do not show Transformer's performance on IMDB because the sequence length is too long for Transformer). On both datasets, SegTree-Transformer outperforms baseline models. On IMDB dataset, our proposed model achieves performance comparable to a bidirectional LSTM initialized with pre-trained character embedding and CoVe embedding (McCann et al., 2017). Compared to Star Transformer, our model is more effective on long text with a 1.25 point gain on IMDB.

4 Related Works

Transformer-XL (Dai et al., 2019) introduces the notion of recurrence into Transformer. It divides the input sequence into multiple segments and recurrently attends to the hidden states of the previous segments. They achieved state-of-the-art on several language modeling benchmarks. Compared to our model, Transformer-XL could only model sequences in one direction, making it hard to deal with tasks where bi-directional information is required.

Shen et al. (2018) proposed a network structured called "bi-directional block self-attention network(Bi-BloSAN)" that splits a sequence into blocks, and sequentially applies inner-block attention and inter-block attention. Compared to their model, our tree-structure has $O(\log n)$ levels while their model has only two levels. In addition to that, by stacking Transformer layers, our model iteratively refines features at all levels, while (Bi-BloSAN) does not have such a mechanism.

5 Reproducibility

The code of our model and hyper-parameter settings with all experiments in this paper would be available at: `https://github.com/yzh119/segtree-transformer-v0`.

References

Ba, Jimmy Lei and Kiros, Jamie Ryan and Hinton, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Battaglia, Peter W and Hamrick, Jessica B and Bapst, Victor and Sanchez-Gonzalez, Alvaro and Zambaldi, Vinicius and Malinowski, Mateusz and Tacchetti, Andrea and Raposo, David and Santoro, Adam and Faulkner, Ryan and others. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Dai, Zihang and Yang, Zhilin and Yang, Yiming and Cohen, William W and Carbonell, Jaime and Le, Quoc V and Salakhutdinov, Ruslan. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

De Berg, Mark and Van Kreveld, Marc and Overmars, Mark and Schwarzkopf, Otfried. Computational geometry. In *Computational geometry*, pp. 1–17. Springer, 1997.

Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Guo, Qipeng and Qiu, Xipeng and Liu, Pengfei and Shao, Yunfan and Xue, Xiangyang and Zhang, Zheng. Star-transformer. *arXiv preprint arXiv:1902.09113*, 2019.

Khandelwal, Urvashi and He, He and Qi, Peng and Jurafsky, Dan. Sharp nearby, fuzzy far away: How neural language models use context. *arXiv preprint arXiv:1805.04623*, 2018.

Li, Jiwei and Luong, Minh-Thang and Jurafsky, Dan and Hovy, Eudard. When are tree structures necessary for deep learning of representations? *arXiv preprint arXiv:1503.00185*, 2015.

Maas, Andrew L and Daly, Raymond E and Pham, Peter T and Huang, Dan and Ng, Andrew Y and Potts, Christopher. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pp. 142–150. Association for Computational Linguistics, 2011.

Marcus, Mitchell and Santorini, Beatrice and Marcinkiewicz, Mary Ann. Building a large annotated corpus of english: The penn treebank. 1993.

McCann, Bryan and Bradbury, James and Xiong, Caiming and Socher, Richard. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pp. 6294–6305, 2017.

Merity, Stephen and Xiong, Caiming and Bradbury, James and Socher, Richard. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Pennington, Jeffrey and Socher, Richard and Manning, Christopher. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Press, Ofir and Wolf, Lior. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Radford, Alec and Narasimhan, Karthik and Salimans, Tim and Sutskever, Ilya. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

Shaw, Peter and Uszkoreit, Jakob and Vaswani, Ashish. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

Shen, Tao and Zhou, Tianyi and Long, Guodong and Jiang, Jing and Zhang, Chengqi. Bi-directional block self-attention for fast and memory-efficient sequence modeling. *arXiv preprint arXiv:1804.00857*, 2018.

Socher, Richard and Perelygin, Alex and Wu, Jean and Chuang, Jason and Manning, Christopher D and Ng, Andrew and Potts, Christopher. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Vaswani, Ashish and Shazeer, Noam and Parmar, Niki and Uszkoreit, Jakob and Jones, Llion and Gomez, Aidan N and Kaiser, Łukasz and Polosukhin, Illia. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Veličković, Petar and Cucurull, Guillem and Casanova, Arantxa and Romero, Adriana and Lio, Pietro and Bengio, Yoshua. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, Minjie and Yu, Lingfan and Gan, Quan and Zheng, Da and Gai, Yu and Ye, Zihao and Li, Mufei and Zhou, Jinjing and Huang, Qi and Zhao, Junbo and Lin, Haibin and Ma, Chao and Deng, Damon and Guo, Qipeng and Zhang, Hao and Li, Jinyang and Smola, Alexander J and Zhang, Zheng. Deep graph library, 2018. URL `http://dgl.ai`.

A APPENDIX

A.1 PSEUDO CODE OF SEGTREE-TRANSFORMER

Algorithm 1 SegTree Transformer

Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the underlying graph, T the number of layers, H^0 initial hidden states

1: **for** i := 1 to T **do**:
2: $\quad Z^i \leftarrow \text{norm}\left(H^{i-1} + \text{GSA}^{(i)}\left(\mathcal{G}, H^{i-1}\right)\right)$
3: $\quad H^i \leftarrow \text{norm}\left(Z^i + \text{FFN}^{(i)}\left(Z^i\right)\right)$
4: **end for**
5: **return** H^T

A.2 RELATIVE POSITION REPRESENTATIONS

As in Shaw et al. (2018), inducing relative distances between words into self-attention computation is helpful. Here we draw a similar analogy on the tree. For each node v in $\mathcal{A}(u)$, we consider the relative positional difference on the tree between u and v, and assign a latent representation $r_{v,u}$ of such difference:

- $r_{v,u} = r_{\text{self}}$ if $v = u$.
- $r_{v,u} = r^j_{\text{left}}$ or r^j_{right}, if v is the left/right node to join the neighbor set of u at the j-th iteration in algorithm(see Figure 1) of finding top-down context nodes.
- $r_{v,u} = r^j_{\text{anc}}$, if u is the ancestor of v in the tree at level j.

All such $r^{\cdot}_{\cdot}$ are trainable parameters.

Then, we modify equation 2.3 to include positional representations:

$$R^u = \text{concat}(\{r_{v,u} \mid v \in \mathcal{A}(u)\}) \qquad head^u_i = \text{softmax}\left(\frac{Q^u_i\,(K^u_i + R^u)^T}{\sqrt{d}}\right) V^u_i \tag{7}$$

Note that the relative positional representations $r^{\cdot}_{\cdot}$ are shared across attention heads(this setting is the same as Shaw et al. (2018)), and each Transformer layer gets its own set of positional representations.

A.3 GPU MEMORY COST

In this section we report the GPU memory cost of SegTree-Transformer on Language Modeling task in two different settings.

Parameter	Setting 1	Setting 2
Number of layers	6	12
Vocabulary size	28913	30522
Hidden size	320	768
FFN hidden size	1024	3072
Batch size	16	8
Optimier	Adam	Adam
Floating-point format	Single-precision	Single-precision

Table 4: Experiment settings

SegTree-Transformer has the same parameter size as Vanilla Transformer, with the property of using less GPU memory.

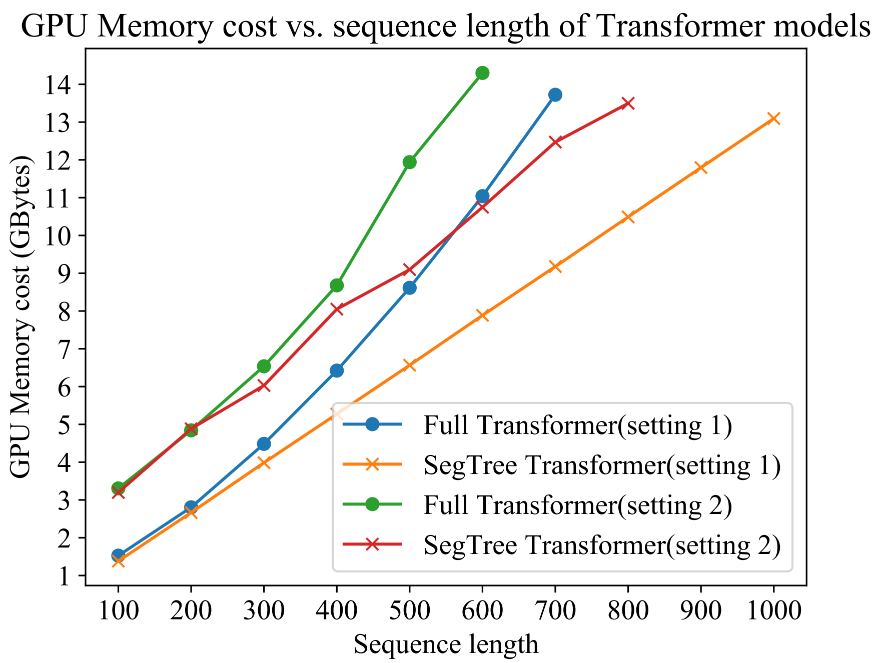


Figure 2: GPU memory vs. sequence length